This document lists the programming standards that you must follow for the programming questions of your assignments. Failure to follow these standards will result in the loss of marks.

Commenting files and methods

- Each of your program files must begin with a comment block like the following:

  /**
   * Name of class or program (matches filename)
   *
   * COMP 2140 SECTION A0?
   * INSTRUCTOR  <your instructor’s name here>
   * ASSIGNMENT  Assignment #, question #
   * @author  your name, your student number
   * @version  date of completion
   *
   * PURPOSE: what is the purpose of your program?
   */

- If the purpose of a method is not self-explanatory, write a prologue comment at the beginning of the method, similar to the following:

  /**
   * PURPOSE: Describe the purpose of the method. Clearly.
   * Indicate how the parameters and return values are used.
   * For instance methods, describe the effect on the object state.
   */

  or:

  /**
   * Describe the purpose of the method. Clearly.
   * For instance methods, describe the effect on the object state.
   *
   * @param firstParameter  description of what firstParameter means
   *                         indicate if the value is modified, and why
   * @param secondParameter  description of what secondParameter means (as above)
   *
   * @return  what is the meaning of the return value?
   */
Writing readable code

- Group the contents of your class consistently. Always place data before behaviours. The following order is recommended:

```java
class Something {
    // Instance variables
    // Static variables
    // Class-level constants
    // Constructors
    // Other instance methods
    // Static methods: main comes first, if it exists
}
```

- Each assignment question should be submitted in a separate Java source code file. Place the main (public) class first and then add any other classes after the main class, as shown below.

```java
public class A2Q1 {
    public static void main( String[] args ) {
        ...
    }
    ...
}
class Worker {
    ...
}
```

- Use blank lines to separate blocks of code and declarations to improve readability. In particular, use blank lines between declarations and other code, and between methods.

- Comment blocks of code. Describe why you wrote the code this way, not what each line does. For example:

```java
// Summarizes taxes in each category (only federal so far)
System.out.println("The taxes payable are:");
System.out.print("Federal tax = ");
System.out.println(federalTax);
```

- Use meaningful but reasonable variable names. Start in lower case and capitalize only the first letter of each new word.

  - a ??? Very bad. Too short, not meaningful.
  - average ??? Good if there is only one possible average
  - averageMark ??? Good
  - average_of_all_the_marks_in_the_list ??? Bad. Too long and wordy.

- Use consistent indentation to clarify control structures (e.g., loops and if constructs). Levels of indentation should clearly indicate the depth of nesting. For example:
class MyClass {
    public static void someMethod() {
        statement;
        while (...) {
            statement;
            if (...) {
                statement;
                ...
            }
            statement;
        }
        statement;
    }
    public static int anotherMethod() {
    }
}

Align else with the corresponding if for readability. Common styles include:

    if (...) {
        statement;
        ...
    } else {
        statement;
        ...
    }

or:

    if (...) {
        statement;
        ...
    } else {
        statement;
        ...
    }

Avoid long lines; where needed, continuations of a statement on a new line should be indented too. Any readable and consistent style is acceptable. The essential features are that all statements that are nested within another statement must be indented, and that the braces {} must be in predictable and consistent positions.
Writing maintainable and extendable code

- Avoid the use of literal constants ("magic numbers") in your program. Generally use constant (final) identifiers rather than literal constants in your program. Acceptable exceptions are strings that appear only once, and small fundamental values. For example:
  - `sum = 0;` ??? Literal constant 0 is OK
  - `count = count + 1;` ??? Literal constant 1 is OK
  - `price = total * 1.07;` ??? Magic values are not proper. Create a named constant like PST_RATE = 1.07.
  - `lastDigit = accountNumber % 10;` ??? Literal constant 10 is OK.
  - `while (!command.equals("Quit")) { ... if (command.equals("Quit")) ...` ??? Duplication is not proper. Create a named constant like QUIT_COMMAND = "Quit".

- Declare variables with as small a scope as possible. Use instance variables for object properties, local variables for method-specific data, and variables declared in `for` loops for counters. Use `static` variables only where absolutely necessary.

- Values of type `float` or `double` should not normally be compared with `==` or `!==`.

- Do not use `break` or `continue` statements (unless you are using a `switch` control structure) without the explicit permission of your instructor.

- Do not use multiple `return` statements in the same method without the explicit permission of your instructor.

- Never change the value of a `for` loop variable inside the loop.

- Use the best possible construct. `For` loops are only used when the number of iterations is known in advance; use a `while` or `do/while` for non-deterministic loops.

- Avoid unnecessary duplication of code. Use methods and inheritance wherever possible to prevent code duplication.

- Use appropriate language-specific naming standards for Java. Class names must start with `InitialCapitalLetters`, constants are `ALL_UPPER_CASE`, and all other identifiers use `initialLowerCase`.

- There should not be any processing performed in the `main` method; `main` should only call the methods that perform the actual processing. (This is also true for constructors.) Thus, the main method will normally consist of several calls to the processing methods. Use meaningful method and variable names. For example, use `printFlights(flights)` instead of `output(f)`.

Input and output

- All output produced on the console must have appropriate titles and headings. Output that is sent to a file typically does not require titles or headings.

- Print console output neatly. Use tabular output where appropriate.

- Print a message on the console at the end of the program that indicates whether or not the program completed successfully; e.g., `End of processing`. This message should be printed by the last statement in your main program.
Classes and Objects

- Never perform any printing inside a `toString()` method. The `toString()` method always returns a `String` that contains information that describes an object.

- Perform as much processing as is possible inside objects. For example, if you have to compare the value of an object’s instance variable with some other value, do not extract the value of the instance variable from the object and compare it with the other value outside of the object; instead, define a method such as `compareValue(double value)` inside the object. The method performs the comparison and returns the result.

- Make your access specifiers as restrictive as possible. Private things should be `private`, including all data members.